数理解析研究所講究録
556巻 1985年 49-58

A Distributed Algorithm for Deadlock Detection in Replicated Database Systems

Masanobu Ogata, Kazuo Sugihara and Tohru Kikuno

Faculty of Engineering, Hiroshima University
Higashi-Hiroshima, 724 JAPAN

1. INTRODUCTION

The deadlock problem in distributed environments has recently received a great deal of attention. Deadlock detection is more suitable than deadlock prevention and avoidance in distributed database systems (DDBSs) [Isloor80, Menasce79]. Furthermore, algorithms for deadlock detection in DDBSs should be distributed ones, since the center node in centralized algorithms may be a bottleneck from the viewpoint of reliability and performance.

A number of distributed algorithms for deadlock detection have been proposed [Badal83, Bracha84, Chandy82, Chandy83, Elmagarmid84, Goldman77, Ho82, Jagannathan82, Kawazu79, Menasce79, Misra82, Mitchell84, Obermarck82, Sugihara84, Tsai82]. These distributed algorithms can be classified into the following three categories with respect to deadlock models used in them.

(1) Resource model: A process can proceed with its execution only when it acquires all the resources that it is waiting for (i.e., an AND request). A deadlock is represented by a cycle in a wait-for graph that represents a system state. Most of the previously proposed algorithms have been devoted to detection of deadlocks in the resource model.

(2) Communication model: A process can proceed only if it receives a message from any one of the processes that it is waiting for (i.e., an OR request). A deadlock is represented by a knot [Chandy83, Misra82] in a wait-for graph.

(3) General model: A process can issue arbitrary requests described by formulas with AND and OR connectives. A deadlock is defined on an AND-OR graph.

A deadlock in the general model may occur in a replicated database system in which copies of each data item may be dispersed over more than one site. A write lock on data item x can be regarded as an AND request for all the copies of x. On the other hand, a read lock on x can be regarded as an OR request for the copies of x. Moreover, some concurrency control schemes such as weighted voting [Gifford81] may require N-out-of-M requests such that a process must acquire at least N copies out of M copies of a data item. Therefore, in order to make deadlock detection independent of concurrency control schemes, deadlocks in replicated database systems should be represented in the general model that allows AND, OR and N-out-of-M requests. Note that AND and OR requests are special cases of the N-out-of-M request in which N=M and N=1, respectively.

Recently, Bracha and Toueg have presented a distributed algorithm for deadlock detection that supports N-out-of-M requests [Bracha84]. The algorithm requires at most 4e messages where e is the number of edges in a wait-for graph G. It provides the best solution with respect to communication complexity among the previously proposed distributed algorithms for deadlock detection in the AND-OR request model as well as that in the N-out-of-M request model. However, it takes 4d hops in the worst case to determine if a process is deadlocked where d is a diameter of G.

This paper discusses distributed deadlock detection in replicated database systems that allow processes to issue any requests described by formulas with AND, OR and N-out-of-M connectives. For example, a process p can request services from processes p_1, p_2,..., p_7 with the following logical condition: p_1 AND 2-out-of-3(p_2,p_3,p_4) AND (p_5 OR p_6 OR p_7). The process p can proceed with its execution only if it is granted the services that satisfies the above formula. However, by parsing the formula and introducing dummy processes, we can assume that any process issues only requests that contain a single N-out-of-M request. Thus, the above request can be formulated as follows: p issues a

3-out-of-3(p_1,q_1,q_2) request, q_1 issues a 2-out-of-3(p_2,p_3,p_4) request and q_2 issues a 1-out-of-3(p_5,p_6,p_7) request, where q_1 and q_2 are dummy processes.

We present a new distributed algorithm for detecting deadlocks in the N-out-of-M request model. The total number of messages required in the algorithm is at most (3e+cn), where e is the number of edges in a wait-for graph G, n is the number of nodes in G and c is the maximum length of simple paths from an initiator of the algorithm in G. The size of each message is at most 2n(log n) bits. Thus, the communication complexity of our algorithm is at most 2n(3e+cn)log n bits, while that of the algorithm in [Bracha84] is at most (4e log n) bits. However, our algotithm takes only at most 3d hops, whereas that in [Bracha84] takes 4d hops in the worst case.

2. THE SYSTEM MODEL

A distributed system is a finite set of processes. A process can communicate with other processes by sending messages to them. Every message sent from u to v is received by v within a finite time and in the order sent (i.e., the First-In-First-Out manner). A process can be either active or blocked. A process is said to be active if it is not waiting for any other process. Otherwise, it is said to be blocked. A process can proceed with its execution only if it is active.

An active process p can issue an N-out-of-M request to M processes by sending REQUEST messages that state for them to carry out a certain action on its behalf. Then, p becomes to be blocked. The process that received a REQUEST message from p can carry out the requested action only when it is active. If it is active, it carries out the action and then sends a REPLY message to p to inform that the action has been completed. When p receives REPLY messages from at least N processes in the M processes, p becomes to be active again. Then, p relinquishes the requests for the rest of the M processes by sending RELINQUISH messages to them.

The global state of a distributed system is represented by a digraph G=(V,E), called a wait-for graph. Each node v∈V corresponds to the process v in the system. A directed edge (v,w)∈E corresponds to a REQUEST message sent from v to w such that v has not received a REPLY message from w and v has not sent a RELINQUISH message to w. Associated with each node v is the number #(v) of REPLYs that v needs to receive to become active. v is said to be active iff #(v)=0.

By OUT(v) we denote the sets of nodes w's such that v sent a REQUEST message to w, and neither v received a REPLY message from w nor v sent a RELINQUISH message to w. That is, OUT(v) is the set of processes that v is waiting for. IN(v) denotes the set of nodes u's such that v received a REQUEST message from u, and neither v sent a REPLY message to u nor v received a RELINQUISH message from u. We assume that each node v knows #(v), OUT(v) and IN(v).

Next, we define transformations T1 and T2 of a wait-for graph G=(V,E) to represent behavior of a distributed system.

T1: Adding edges from any active node v∈V to r nodes in V and setting #(v) to some k ($1 \leq k \leq r$).

T2: Deleting an edge (v,w)∈E and decreasing #(v) by 1. If #(v)=0, then all the outgoing edges of v are deleted.

The transformation T1 corresponds to change of a system state when v issues an N-out-of-M request where N=k and M=r. The transformation T2 corresponds to change of the system state when v receives a REPLY message from w or v sends a RELINQUISH message to w.

If application of a transformation s to G results in a wait-for graph G', we write $G|\overset{s}{---}G'$. If $G_0|\overset{s_1}{---}G_1$, $G_1|\overset{s_2}{---}G_2$, ..., $G_{k-1}|\overset{s_k}{---}G_k$, then we denote $G_0|\overset{\sigma}{---}G_k$ for the sequence $\sigma = s_1 s_2 \ldots s_k$. For a given G, a sequence σ of transformations is said to be a schedule σ for G if $G|\overset{\sigma}{---}G'$ for some wait-for graph G'. A node v is deadlocked in G iff there is no schedule

σ for G such that $G|\overset{\sigma}{\text{---}}G'$ and v is active in G'. Otherwise, v is live.

3. DISTRIBUTED DEADLOCK DETECTION

We first present a key idea of our distributed algorithm that is the two-phase algorithm (see APPENDIX). It consists of the tree construction phase and the activation phase. Let p be the node, called an initiator, that initiates the algorithm to determine if p is deadlocked.

In the tree construction phase, p finds a set REACH of all nodes v such that there is a path from p to v and also computes a set ACTIVE of all edges incoming to active nodes in REACH. Simultaneously, it constructs a directed tree T with the root p such that T includes all the nodes in REACH. This phase can be executed by using the echo algorithm in [Chang82].

In the activation phase, p informs every node in REACH to start the activation phase by sending START messages along edges in the tree T. Let father and CHILD(v) be the father of v and the set of all children of v in T. Then, every active node v$\in$REACH sends ACTIVATE messages to all the nodes in IN(v) to search for live nodes in REACH. The ACTIVATE messages propagate in a forest-like pattern on G. When an ACTIVATE message to a node v does not change the Boolean variable "live" of v which indicates if v is live, v sends its father a DONE message to inform p that the propagation of this ACTIVATE message terminates. The algorithm terminates when the propagation of every ACTIVATE message terminates. Then, p is deadlocked iff "live" of p is false.

The major difference between the distributed algorithm in [Bracha84] and the two-phase algorithm is how an initiator p knows the termination of searches for live nodes in REACH. In our algorithm, p maintains two sets SEARCH and TERM to know when the propagation of every ACTIVATE message terminates. SEARCH is a set of the edges that an ACTIVATE message has been sent along. It is initially equal to ACTIVE. TERM is a set of edges that an ACTIVATE message has already passed through.

Thus, the activation phase terminates iff SEARCH=TERM. To maintain these sets consistently, each ACTIVATE(X,Y) message includes two sets X and Y that represent partial information about SEARCH and TERM, respectively. On the other hand, in the algorithm in [Bracha84], each active node first knows the termination of search initiated by itself by using "echoes" [Chang82] and then p is informed the termination of every search.

Next, we present a sketch of our distributed algorithm in which the tree construction phase and activation phase in the two-phase algorithm are executed in parallel. That is, these phases are synchronized so that each node executes the former before the latter. An active node sends ACTIVATE messages to start the activation phase when it has completed its execution of the tree construction phase. Thus, there is no need of START messages. When an ACTIVATE message arrives at a node that has not executed the tree construction, the message awaits that the node completes it. If ACTIVATE arrives at a node that is not reachable from p, the node waits for a message of the tree construction phase forever. Thus, after p determines if p is deadlocked, p has to send such nodes TERMINATE messages that inform them the termination of deadlock detection.

4. CORRECTNESS AND PERFORMANCE

It is clear that when every search for live nodes terminates, an initiator p is deadlocked iff live of p is false. Thus, we show only that after the termination of every search, p knows the termination of deadlock detection within a finite time. For each ACTIVATE(X,Y), X is a subset of Y. Thus, TERM is always a subset of SEARCH. Note that SEARCH=ACTIVE and TERM=ϕ when the activation phase starts. Therefore, TERM is a proper subset of SEARCH so far as there is an ACTIVATE message in a system. On the other hand, any edge that an ACTIVATE message has passed through is always included in X of at least one message ACTIVATE(X,Y). If ACTIVATE(X,Y) has terminated, X is added to TERM within a finite time by a DONE message. Thus, after every ACTIVATE message terminates, TERM will become equal

to SEARCH (i.e., p knows the termination of deadlock detection) within a finite time.

Next, we analyze the communication, time and space complexities of our distributed algorithm. Let n and e be the number of nodes and edges, respectively. Let c and d be the maximum length of simple paths from an initiator in G and a diameter of G. The algorithm requires at most 2e messages in the tree construction phase, and at most e ACTIVATE messages, at most (c-1)(n-1) DONE messages and at most (n-1) TERMINATE messages in the activation phase. Thus, the total number of messages is at most (3e+cn). Since the size of each message is at most (2n log n) bits, the total amount of communication is at most 2n(3e+cn)log n bits. Suppose that all the message transmissions are synchronized and take one unit of time, called a hop, and local computation time is negligible. The two-phase algorithm requires 2d and 3d hops to execute the tree construction phase and the activation phase, respectively. However, our algorithm requires only at most 3d hops, since these two phases can be executed in parallel. Thus, it provides the better solution with respect to the time complexity than the algorithm in [Bracha84]. It also requires at most (2kn log n) bits of local storage where $k=\max_{v}\{|OUT(v)|\}$.

5. CONCLUDING REMARKS

In this paper, we have presented the new distributed algorithm for detecting deadlocks in the N-out-of-M request model. It determines if its initiator is deadlocked in a "static" situation where a system state does not change. However, it is easy to extend the algorithm to a "dynamic" situation where the system state dynamically changes, by taking a "snapshot" of a state of each process in the same way as the algorithm in [Bracha84].

ACKNOWLEDGEMENT

We are grateful to Professor Noriyoshi Yoshida for his encouragement and advice. We also appreciate Mr. Mototaka Ogino for his help on preparation of the manuscript.

REFERENCES

[Badal83] Badal, D. Z. and Gehl, M. T.:"On deadlock detection in distributed computing systems," Proc. INFOCOM'83, pp. 36-45 (1983).

[Bracha84] Bracha, G. and Toueg, S.:"A distributed algorithm for generalized deadlock detection," Proc. 3rd ACM Symp. on Principles of Distributed Computing, pp. 285-301 (1984).

[Chandy82] Chandy, K. M. and Misra, J.:"A distributed algorithm for detecting resource deadlocks in distributed systems," Proc. ACM Symp. on Principles of Distributed Computing, pp. 157-164 (1982).

[Chandy83] Chandy, K. M., Haas, L. M. and Misra, J.:"Distributed deadlock detection," ACM Trans. Computer Systems, 1, 2, pp. 144-156 (1983).

[Chang82] Chang, E. J. H.:"Echo algorithms: Depth parallel operations on general graphs," IEEE Trans. Software Eng., SE-8, 4, pp. 391-401 (1982).

[Elmagarmid84] Elmagarmid, A. K., Datta, A. K. and Liu, M. T.:"Distributed deadlock detection algorithm in transaction-processing systems," Proc. COMPSAC'84, pp. 81-90 (1984).

[Gifford81] Gifford, D. K.:"Violet, an experimental distributed systems," Computer Networks, 5, 6, pp. 423-433 (1981).

[Goldman77] Goldman, B.:"Deadlock problem in computer networks," Tech. Rep. MIT/LCS/TR-185, M.I.T., Cambridge, Mass. (1977).

[Ho82] Ho, G. S. and Ramamoorthy, C. V.:"Protocols for deadlock detection in distributed database systems," IEEE Trans. Software Eng., SE-8, 6, pp. 554-557 (1982).

[Isloor80] Isloor, S. S. and Marsland T. A.:"The deadlock problem: An overview," Computer, 13, 9, pp. 58-78 (1980).

[Jagannathan82] Jagannathan, J. R. and Vasudevan, R.:"A distributed deadlock detection and resolution scheme: Performance study," Proc. 3rd Int'l Conference on Distributed Computing Systems, pp. 496-501 (1982).

[Kawazu79] Kawazu, S. Minami, S., Itoh, K. and Teranaka, K.: "Two-phase deadlock detection algorithm in distributed databases," Proc. 5th VLDB, pp. 360-367 (1979).

[Menasce79] Menasce, D. A. and Muntz, R. R.:"Locking and deadlock detection in distributed database," IEEE Trans. Software Eng., SE-5, 3, pp. 195-202 (1979).

[Misra82] Misra, J. and Chandy, K. M.:"A distributed graph algorithm: Knot detection," ACM Trans. Programming Language and Systems, 4, 4, pp. 678-686 (1982).

[Mitchell84] Mitchell, D. P. and Merritt, M. J.:"A distributed algorithm for deadlock detection and resolution," Proc. 3rd ACM Symp. on Principles of Distributed Computing, pp. 282-284 (1984).

[Obermarck82] Obermarck, R.:"Distributed deadlock detection algorithm," ACM Trans. Database Systems, 7, 2, pp. 187-208 (1982).

[Sugihara84] Sugihara, K., Kikuno, T., Yoshida, N. and Ogata, M.:"A distributed algorithm for deadlock detection and resolution," Proc. 4th Symp. on Reliability in Distributed Software and Database Systems, pp. 169-176 (1984).

[Tsai82] Tsai, W.-C. and Belford, G. G.:"Detecting deadlock in a distributed system," Proc. INFOCOM'82, pp. 89-95 (1982).

APPENDIX

Two-Phase Algorithm for Node v;

```
begin
  /* Tree Construction */
  compute REACH and ACTIVE and construct a tree T;
  /* Activation */
  if ACTIVE =ø then declare "v is deadlocked"
  else begin
    the initiator p sends START to all w ∈ CHILD(v);
    SEARCH := ACTIVE; TERM := ø; live := false for each node;
```

```
    Upon receipt by v of START:
      if OUT(v) = ∅ then
        begin
          live := true;
          for each w ∈ IN(v) do
            send ACTIVATE({(w,v)},{(u,v)|u∈IN(v)}) to w;
        end
      else for each w ∈ CHILD(v) do send START to w;
    Upon receipt by v of ACTIVATE(X,Y):
      if v is a node in T then
        begin
          #active := #active + 1;
          if ¬live and #active ≥ #(v) then
            begin
              live := true;
              if v ≠ p then
                for each w ∈ IN(v) do
                  send ACTIVATE(X ∪{(w,v)},Y ∪{(u,v)|u∈IN(v)})
                                                          to w;
            end
          else if v ≠ p then send DONE(X,Y) to father;
          if p = v then begin
                          TERM := TERM ∪ X;
                          SEARCH := SEARCH ∪ {(u,w)∈Y|u∈REACH}
                        end;
        end;
    Upon receipt by v of DONE(X,Y):
      if v = p then
        begin
          TERM := TERM ∪ X;
          SEARCH := SEARCH ∪{(u,w)∈Y|u∈REACH};
          if TERM = SEARCH then
            if live then declare "v is not deadlocked"
                    else declare "v is deadlocked";
        end
      else send DONE(X,Y) to father;
  end
end.
```